Efficient Subgraph Matching by Postponing Cartesian Products

Never Stand Still | Faculty of Engineering | Computer Science and Engineering

Lijun Chang

Lijun.Chang@unsw.edu.au
The University of New South Wales, Australia

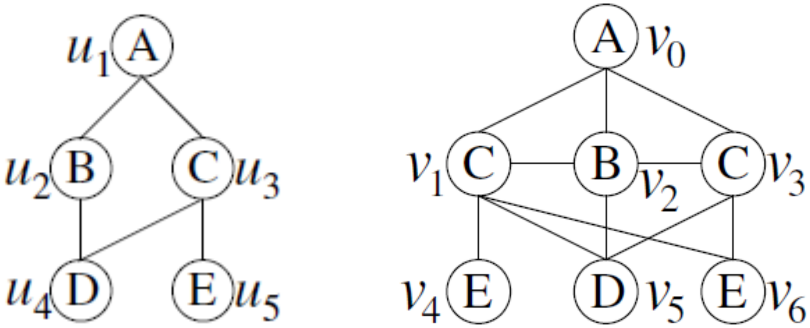
Outline

- Introduction & Existing Works
- Challenges of Subgraph Matching
- Our Approach: CFL-Match
  - Core-First based Framework
  - Compact Path Index (CPI) based Matching
- Experiment
- Conclusion

Introduction

- **Subgraph Matching**

  Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.

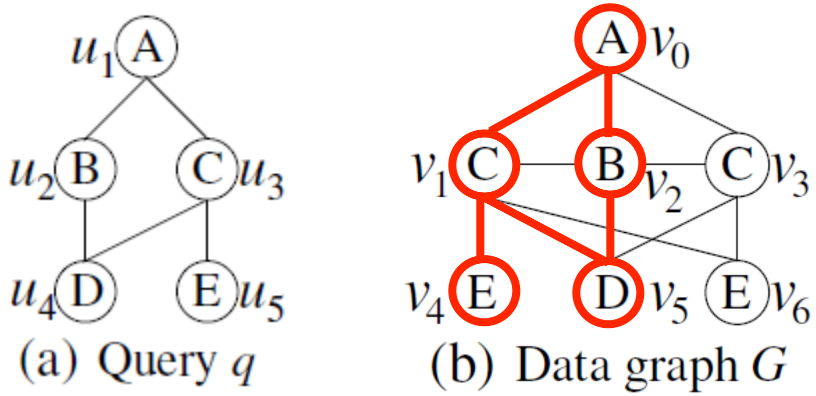


(a) Query q (b) Data graph G

Introduction

➢ **Subgraph Matching**

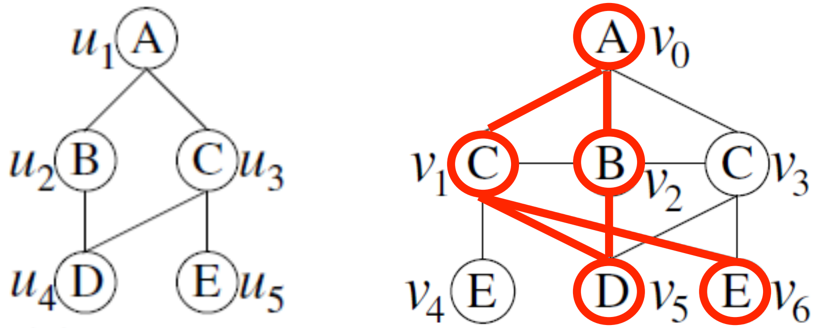
Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.



Introduction

- **Subgraph Matching**

  Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.

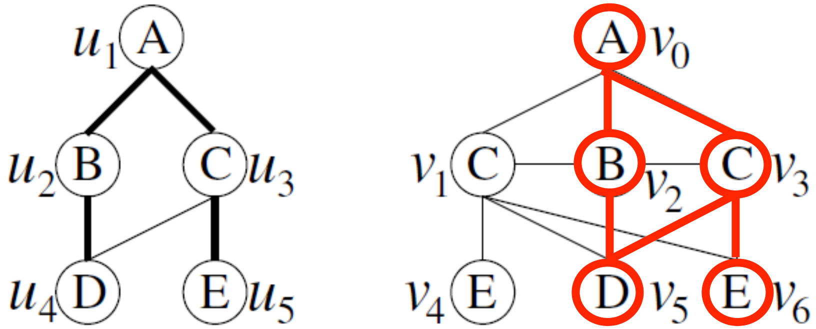


(a) Query q (b) Data graph G

Introduction

➢ **Subgraph Matching**

Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.

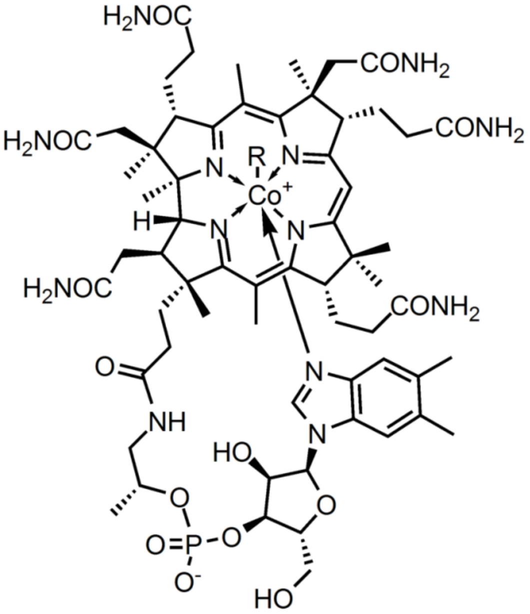


(a) Query q (b) Data graph G

THE UNIVERSITY OF NEW SOUTH WALES

Introduction

- **Applications**
  - Protein interaction network analysis
  - Social network analysis
  - Chemical compound search

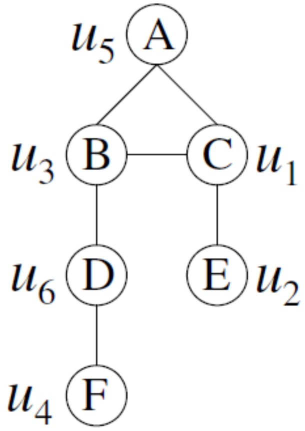


Hardness Result

- Subgraph Isomorphism Testing
  - Decide whether there is a subgraph of ***G*** that is isomophic to ***q***
  - NP-complete
- Enumerating all subgraph embeddings is harder
  - This is the problem we study

Existing Work

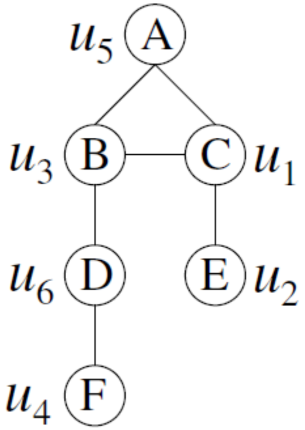
- **Ullmann's algorithm [J.ACM'76]**
  - Iteratively maps query vertices one by one, following the input order of query vertices.
  - Example: Input order could be (u_1, u_2, u_3, u_4, u_5, u_6)
  - Cartesian Products between vertices' candidates.



- **VF2 [IEEE Trans'04] and QuickSI [VLDB'08]**
- **Turbo$_{ISO}$ [SIGMOD'13]**
- **Boost$_{ISO}$ [VLDB'15]**

Existing Work

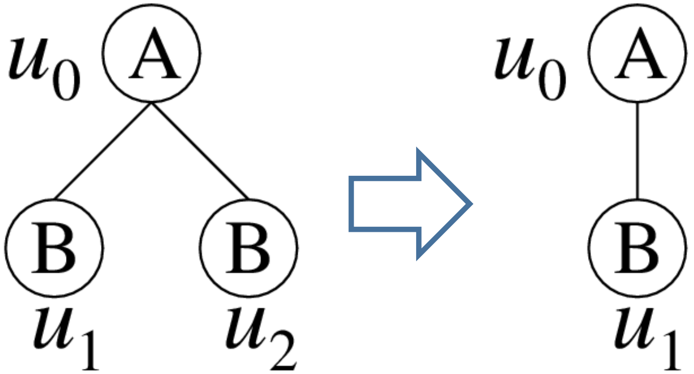
- **Ullmann's algorithm [J.ACM'76]**
- **VF2 [IEEE Trans'04] and QuickSI [VLDB'08]**
  - Independently propose to enforce **connectivity** of the matching order to reduce Cartesian products caused by disconnected query vertices.
  - QuickSI further removes false-positive candidates by first processing infrequent query vertices and edges.
  - Connected order could be (u_5, u_1, u_2, u_3, u_6, u_4)



- **Turbo$_{ISO}$ [SIGMOD'13]**
- **Boost$_{ISO}$ [VLDB'15]**

Existing Work

- **Ullmann's algorithm [J.ACM'76]**
- **VF2 [IEEE Trans'04] and QuickSI [VLDB'08]**
- **Turbo$_{\text{ISO}}$ [SIGMOD'13]**
  - Merge together query vertices with **the same neighborhood**.
    - Reduces Cartesian product caused by similar query vertices
  - Build a data structure online to facilitate the search process.
- **Boost$_{\text{ISO}}$ [VLDB'15]**



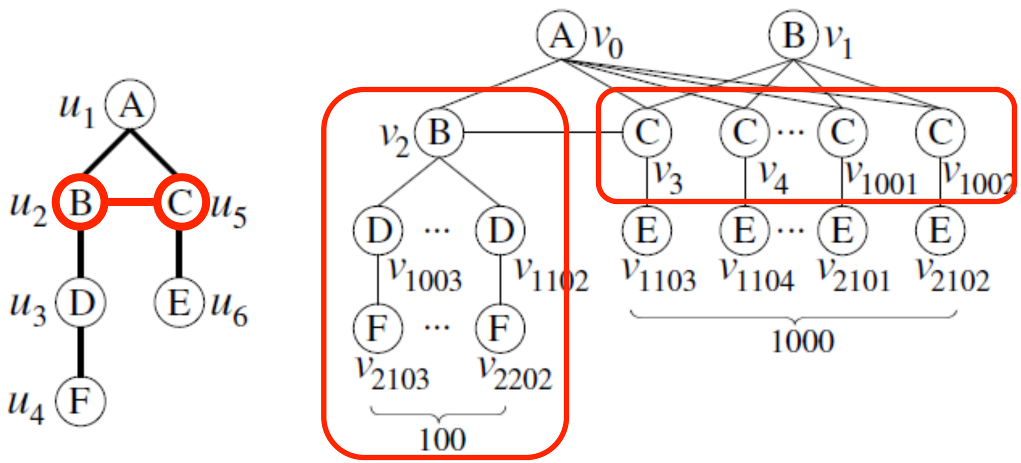
Existing Work

- **Ullmann's algorithm [J.ACM'76]**
- **VF2 [IEEE Trans'04] and QuickSI [VLDB'08]**
- **Turbo$_{\text{ISO}}$ [SIGMOD'13]**
- **Boost$_{\text{ISO}}$ [VLDB'15]**
  - Compress a data graph ***G*** by merging together **similar vertices in *G***.
  - Develop **query-dependent** relationship between vertices in ***G***.

It is still challenging for matching large query graphs.

Challenges of Subgraph Matching

Challenge I: Redundant Cartesian Products by Dissimilar Vertices.



(a) Query q (b) Data graph G

10^5 - 100 partial mappings are redundant.

Matching order of QuickSI and Turbo$_{ISO}$: (u_1, u_2, u_3, u_4, u_5, u_6).

(u_1, u_2, u_5, u_3, u_4, u_6)

Cartesian products:

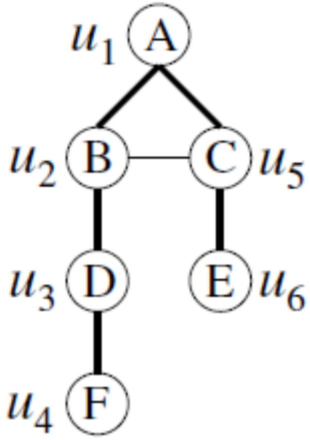
- 100 mappings (v_0, v_2, v_{1000+i}, v_{2100+i}) ($3 \leq i \leq 102$) of (u_1, u_2, u_3, u_4)
- 1000 mappings (v_0, v_j) ($3 \leq j \leq 1002$) of (u_1, u_5)

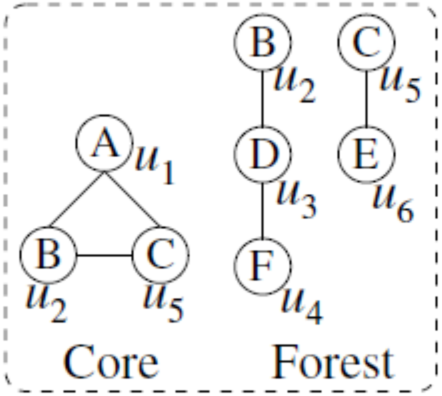
Challenges of Subgraph Matching

Challenge I: Redundant Cartesian Products by Dissimilar Vertices.

Our Solution : Postpone Cartesian products.

- Decompose ***q*** into **a dense subgraph** and **a forest**, and process the dense subgraph first.





Challenges of Subgraph Matching

Challenge II: Exponential size of the path-based data structure in $\textbf{Turbo}_{\textbf{ISO}}$.

- $\text{Turbo}_{\text{ISO}}$ builds a data structure that materializes all embeddings of query paths in a data graph
  1. for generating matching order based on estimation of #candidates.
  2. for enumerating subgraph isomorphic embeddings.

- Worst-case space complexity: $\mathbf{O(|V(G)|^{|v(q)-1|})}$.

UNSW THE UNIVERSITY OF NEW SOUTH WALES

Challenges of Subgraph Matching

Challenge II: Exponential size of the path-based data structure in Turbo$_{\text{ISO}}$.

Our Solution: Polynomial-size data structure, compact path-index (CPI) .

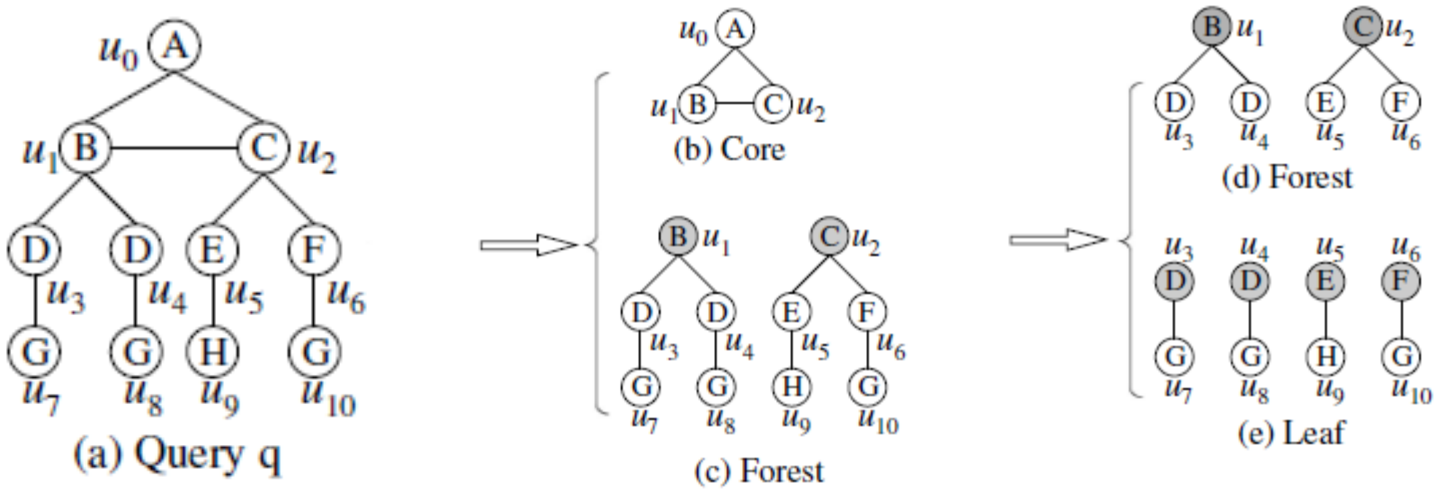
Our Approach

- CFL-Match
  - A Core-First based Framework
  - Compact Path-Index (CPI) based Matching

CFL-Match

- **A Core-First based Framework**
  - Core-Forest Decomposition

    Compute the **minimal connected** subgraph containing **all non-tree edges** of ***q*** regarding any spanning tree.

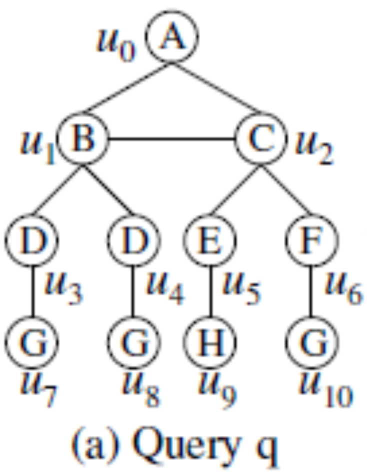


  - Forest-Leaf Decomposition

    Compute the set of **leaf vertices** by rooting each tree at its connection vertex.

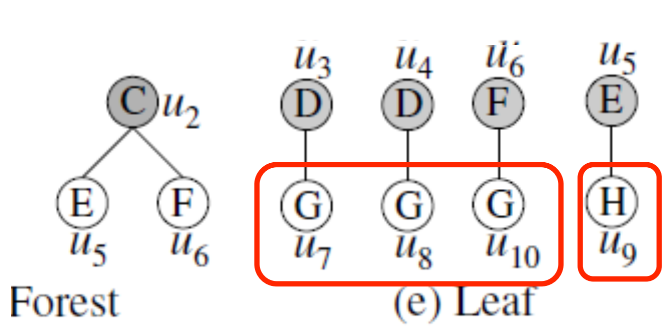
CFL-Match

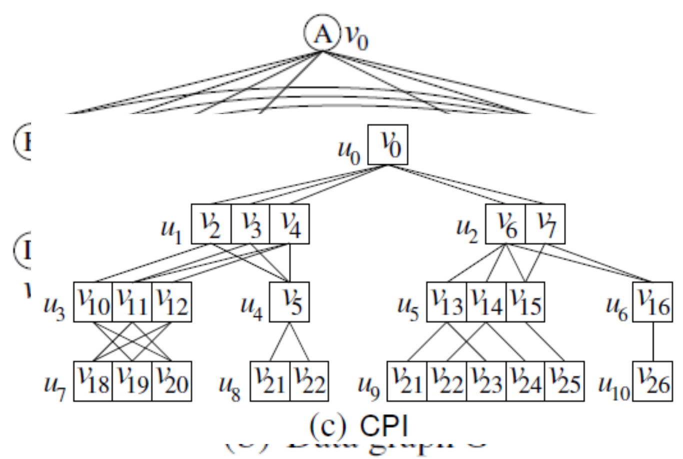
- **A Core-First based Framework**
  1) Core-Forest-Leaf Decomposition
  2) CPI Construction
  3) Mapping Extraction
     i. Core-Match
     ii. Forest-Match
     iii. Leaf-Match



gorize leaf nodes according to labe

ɔrm combination instead of enumer





Auxiliary Data Structure

- **Compact Path-Index (CPI)**
  - Compactly store candidate embeddings of query spanning trees.
  - Serve for computing an effective matching order.

- **CPI Structure**
  - **Candidate sets**

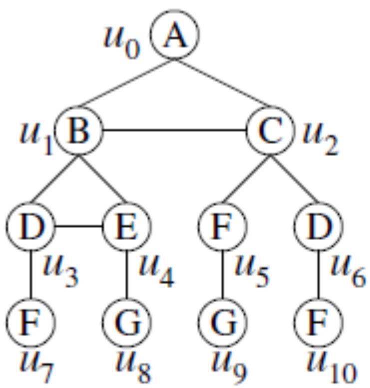
    Each query node ***u*** has a candidate set ***u.C***.
  - **Edge sets**

    This is an edge between ***v*** $\in$ ***u.C*** and ***v'*** $\in$ ***u'.C*** for adjacent query nodes ***u*** and ***u'*** in CPI if and only if ***(v, v')*** exists in ***G***.

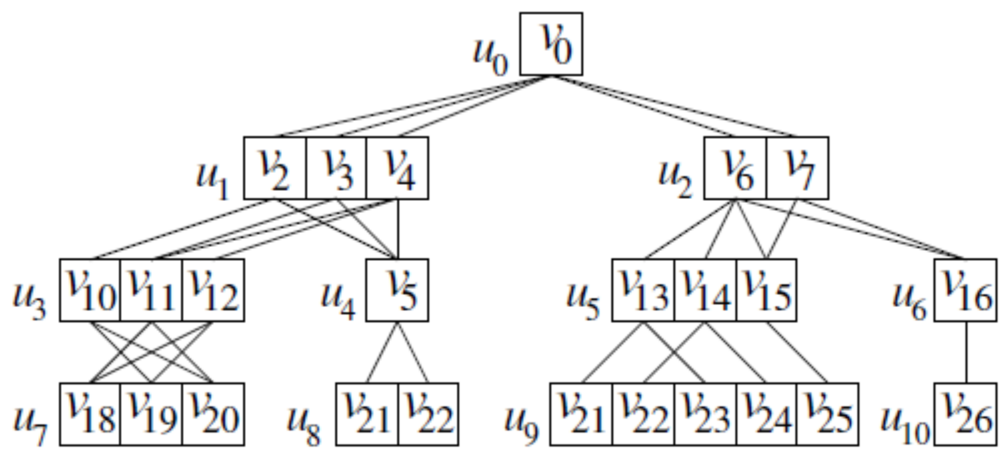
Auxiliary Data Structure

- **Compact Path-Index (CPI)**
  - Compactly store candidate embeddings of query spanning trees.
  - Serve for computing an effective matching order.

- **CPI Structure**
  - **Example**



(a) Query q



(c) CPI

Auxiliary Data Structure

- **Soundness of CPI**

  For every query node u in CPI, if there is an embedding of q in G that maps u to v, then v must be in $u.C$.

Theorem

Given a sound CPI, all embeddings of q in G can be computed by **traversing only the CPI** while G is only probed for non-tree edge checkings.

- **It is NP-hard to build a minimum sound CPI.**

- **Aim to build a small and sound CPI**.

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

CPI Construction

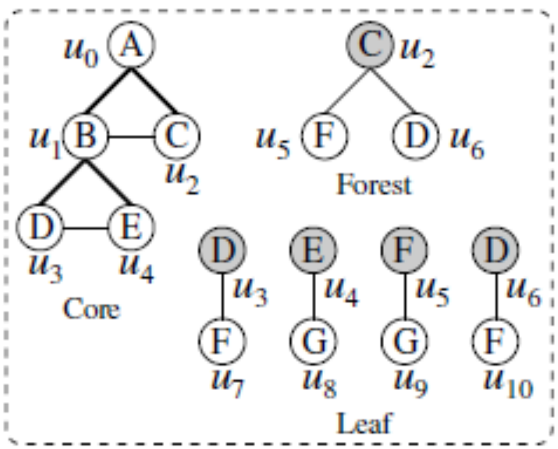
- **General Idea**
  - A heuristic approach:
    1) ***u.C*** is initialized to contain all vertices in ***G*** with the same label as ***u***
    2) A data vertex ***v*** is pruned from ***u.C*** ,

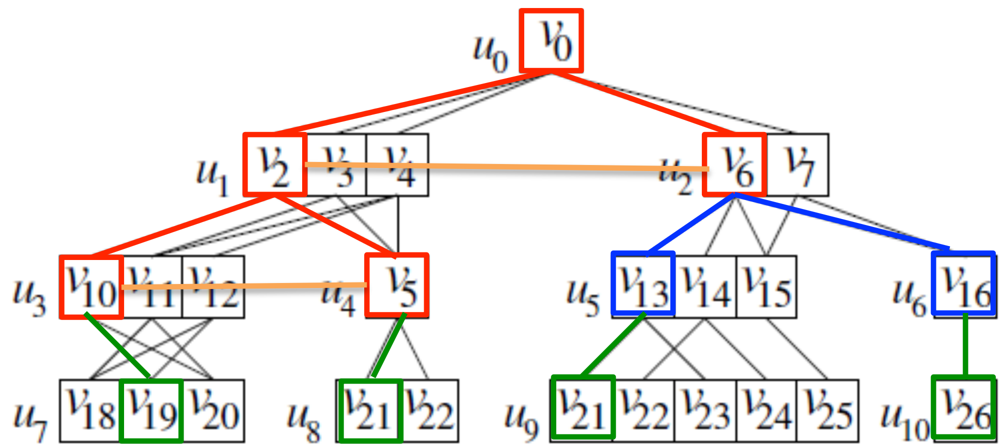
       if $\exists\, u' \in N_q(u)$, such that $\nexists v' \in N_G(v)\ \&\ v' \in u'.C$.

- **A two-phase CPI construction process:**
  - Top-down construction, bottom-up refinement
  - Exploit the pruning power of both directions of every query edge.
  - Construct CPI of **O(|E(G)| X |V(q)|)** size in **O(|E(G)| X |E(q)|)** time

CPI-based Match

- **Compute path-based matching order using CPI**
  - Estimate #matches for each root-to-leaf path in CPI
  - Add paths to the matching order in increasing order regarding #matches
- **Traverse CPI to find mappings for query vertices**
  - Only probe ***G*** for non-tree edge validation





(c) CPI

$(u_0, u_1, u_4, u_3, u_2, u_5, u_6, u_7, u_8, u_9, u_{10})$

Experiment

- All algorithms are implemented in C++ and run on a machine with 3.2G CPU and 8G RAM.
- **Datasets**
  - Real Graphs

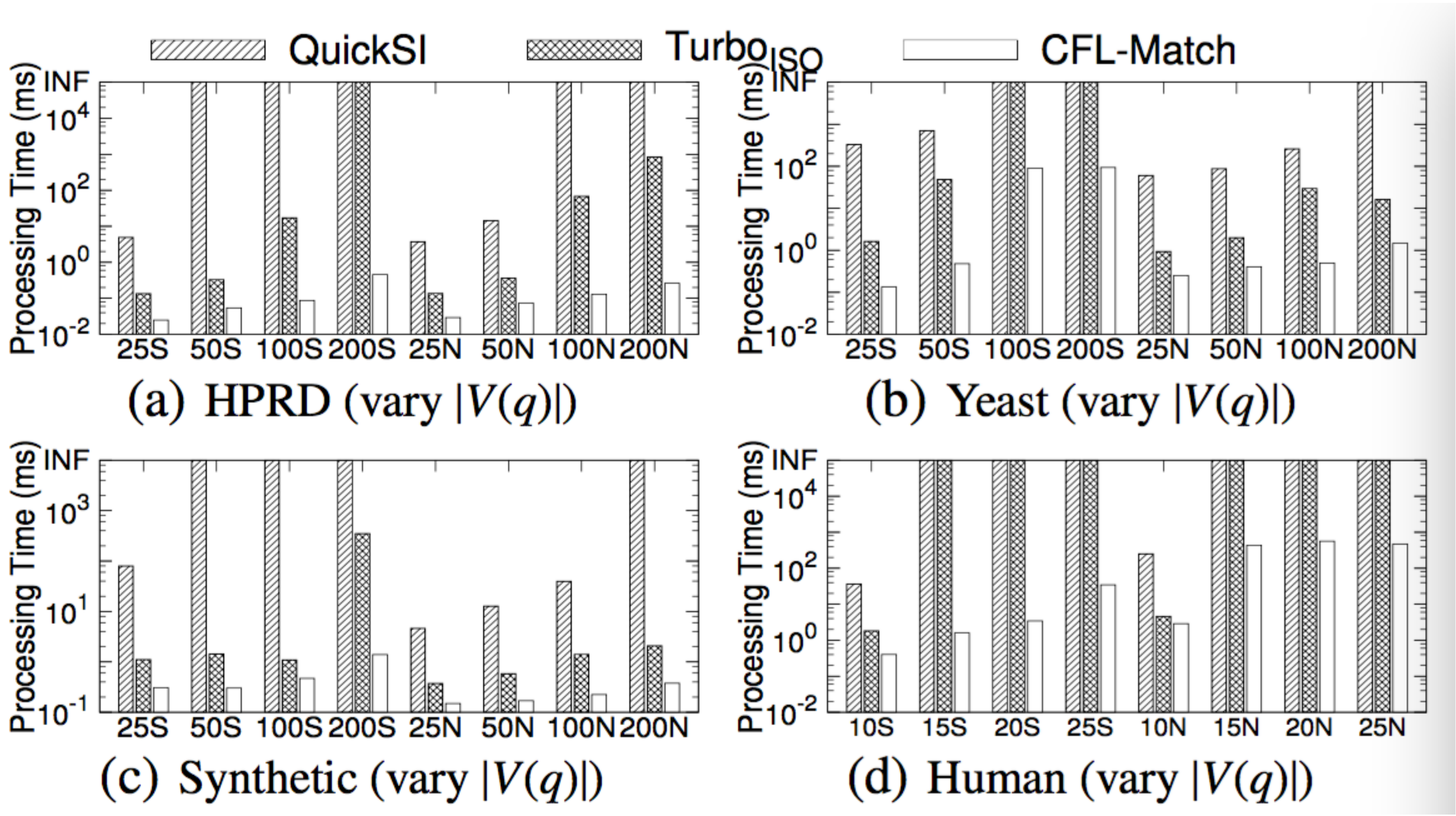
	|V|	|E|	|∑|	Degree
HPRD	9460	37081	307	7.8
Yeast	3112	12519	71	8.1
Human	4674	86282	44	36.9

  - Synthetic Graphs
    - Randomly generate graphs with 100k vertices with average degree 8 and 50 distinct labels.
- **Query Graphs**
  - Randomly generate by random walk
  - Two Categories:

    S: sparse (average degree ≤ 3). N: non-sparse (average degree > 3).

Comparing with Existing Techniques

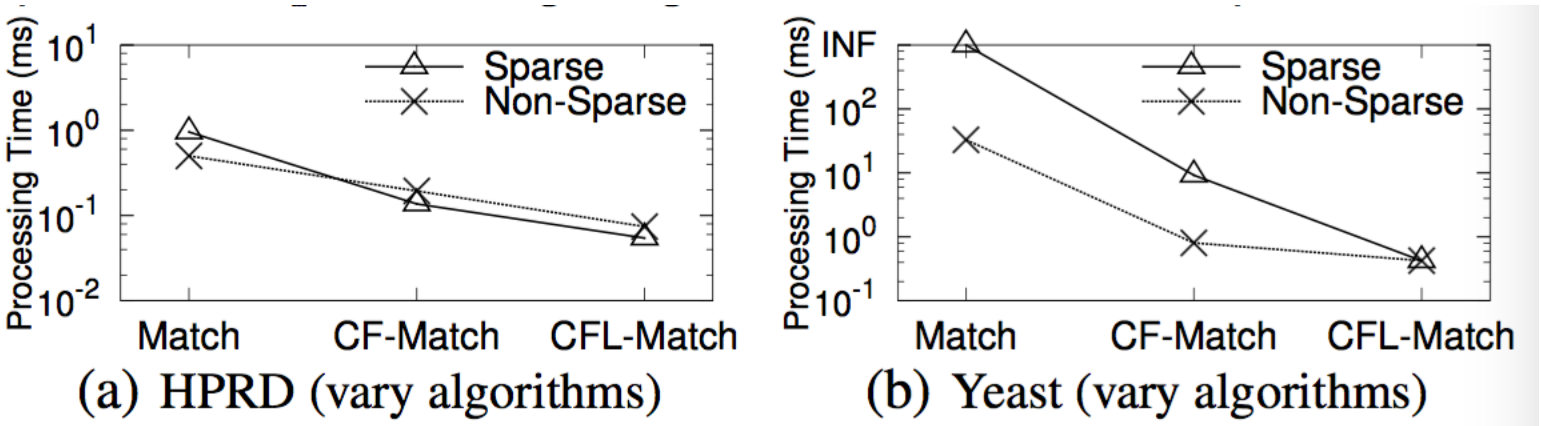
CFL-Match: our proposed algorithm



Varying the size of query graph |V(q)|

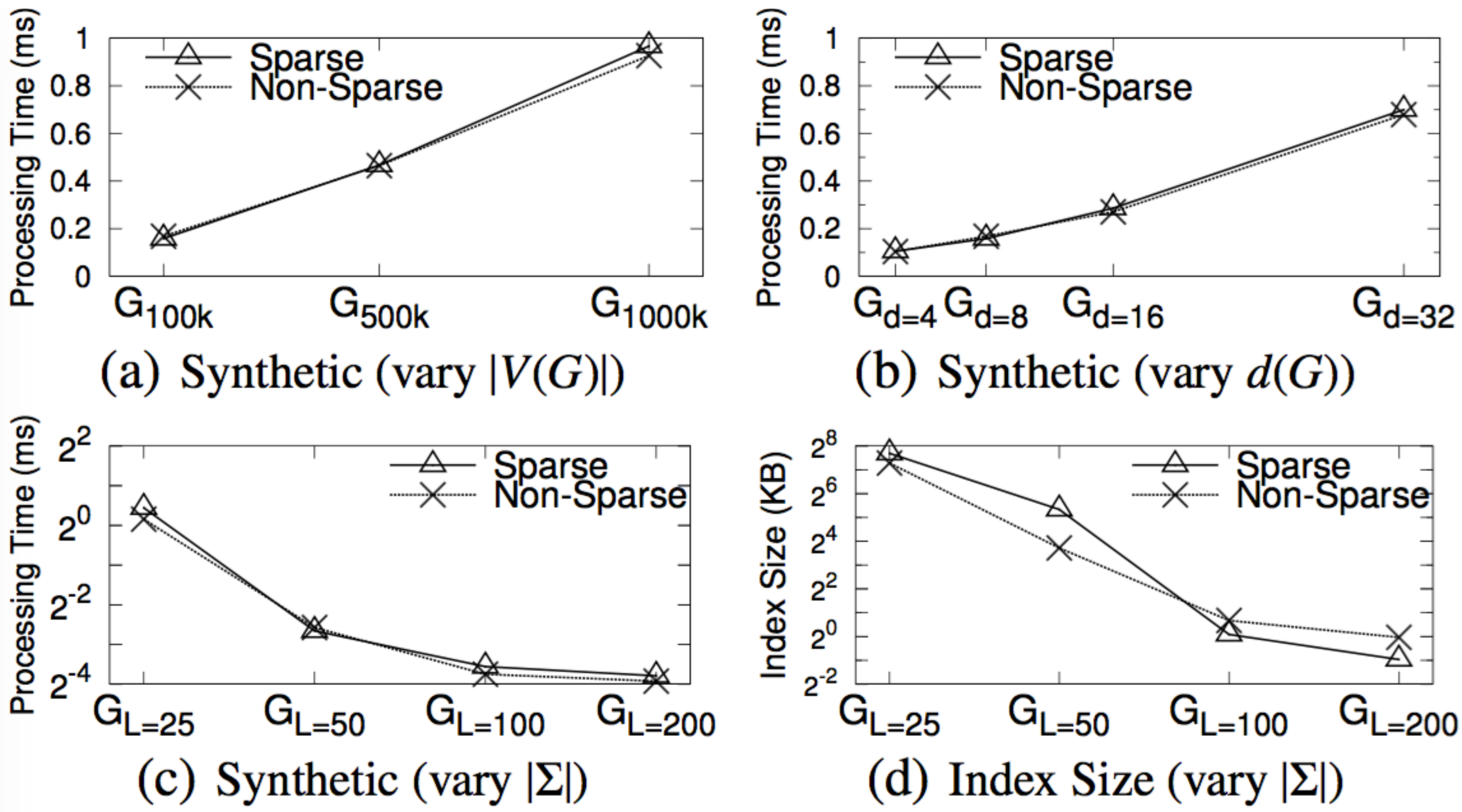
Effectiveness of Our New Framework

- Match: subgraph matching algorithm with CPI but no query decomposition.
- CF-Match: only core-forest decomposition with CPI.
- CFL-Match: our best algorithm.



Evaluating our framework

Scalability Testing



(a) Synthetic (vary $|V(G)|$)

(b) Synthetic (vary $d(G)$)

(c) Synthetic (vary $|\Sigma|$)

(d) Index Size (vary $|\Sigma|$)

Conclusion

- We proposed a core-first framework for subgraph matching by postponing Cartesian products
- We proposed a new polynomial-size path-based auxiliary data structure CPI, and proposed efficient and effective technique for constructing a small CPI
- We proposed efficient algorithms for subgraph matching based on the core-first framework and the CPI
- Extensive empirical studies on real and synthetic graphs demonstrate that our technique outperforms the state-of-the-art algorithms.

Thank you!

Questions?

Lijun.Chang@unsw.edu.au